Program 8 - Bulldog Strategy

Author: Abigail Pitcairn
Version: April 16, 2025

Code: https://github.com/abbyPitcairn/BullDog

Task 1: Creating a Class to Encapsulate Game Status

I began by giving the AI the BulldogGUI and Referee classes and the instructions for task 1. It gave me the new GameStatus class, which I copy/pasted into Eclipse. It then asked if I wanted to integrate these changes into Referee and BulldogGUI, which I confirmed I wanted and then I replaced the Referee class with ChatGPT's version and added some methods it gave me for BulldogGUI. However, this ended up breaking the game functionality.

I gave ChatGPT all of the contents of all of the classes from the program and asked it to fix the broken functionality. It told me the code "looked great" and then gave suggestions for improvements, rather than identifying the actual bugs or offering solutions. One interesting note it gave when I asked it to fix my code:

> "**GUI Responsiveness**:
>
> - If you later add delays or animations (like showing dice rolls), you may want to run logic-heavy operations off the EDT (Event Dispatch Thread) to prevent freezing. For now it's fine, just a heads-up for future-proofing."

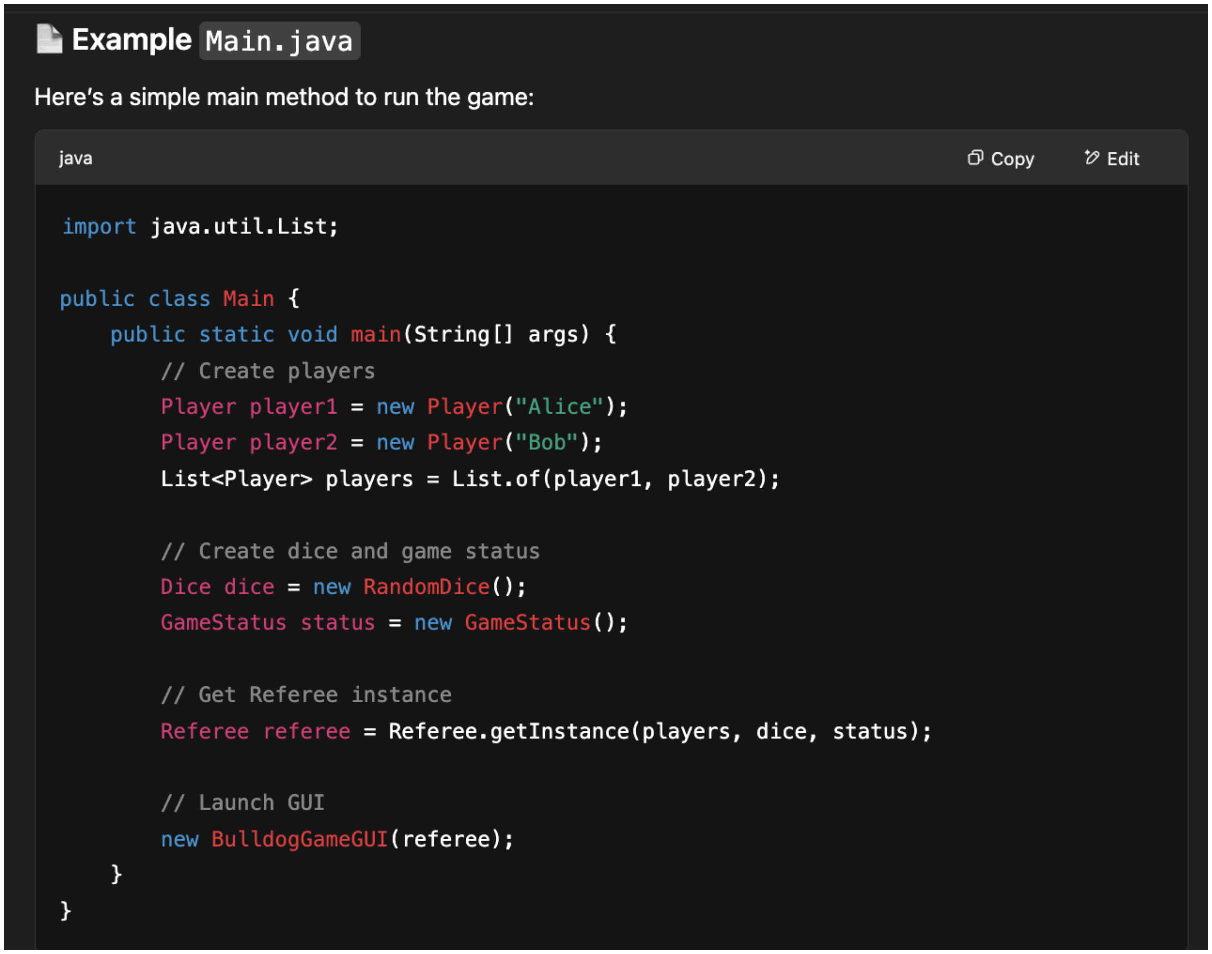
I thought this reply was interesting because I had not mentioned anything like this before (this was a totally new chat that had no previous messages in it) so the AI came up with this idea on its own and wanted to give me a "heads-up".

I clearly needed to take a step back and try again. I edited my previous prompt to simply ask if ChatGPT understood the code. It said yes, explained it, and offered some improvements. It's improvements included:

🔄 **Decouple GUI from game logic more cleanly**: GUI still directly calls `endTurn()` and `nextTurn()` internally. Ideally, only `Referee` should manage turns.

🎯 **Move `rollDice()` and `endTurn()` fully to `Referee`**: Right now, they're duplicated in both `GUI` and `Referee`.

I liked these suggestions and asked ChatGPT to implement them. Technically, I was not yet working on task 1, but this felt important to do first. However, ChatGPT was simply unable to implement the game logic correctly. In its edits, it removed the main() method entirely. I prompted it, "how do I run the program?" and it gave me a main() method that looked like this:



Example `Main.java`

Here's a simple main method to run the game:

```java
import java.util.List;

public class Main {
    public static void main(String[] args) {
        // Create players
        Player player1 = new Player("Alice");
        Player player2 = new Player("Bob");
        List<Player> players = List.of(player1, player2);

        // Create dice and game status
        Dice dice = new RandomDice();
        GameStatus status = new GameStatus();

        // Get Referee instance
        Referee referee = Referee.getInstance(players, dice, status);

        // Launch GUI
        new BulldogGameGUI(referee);
    }
}
```

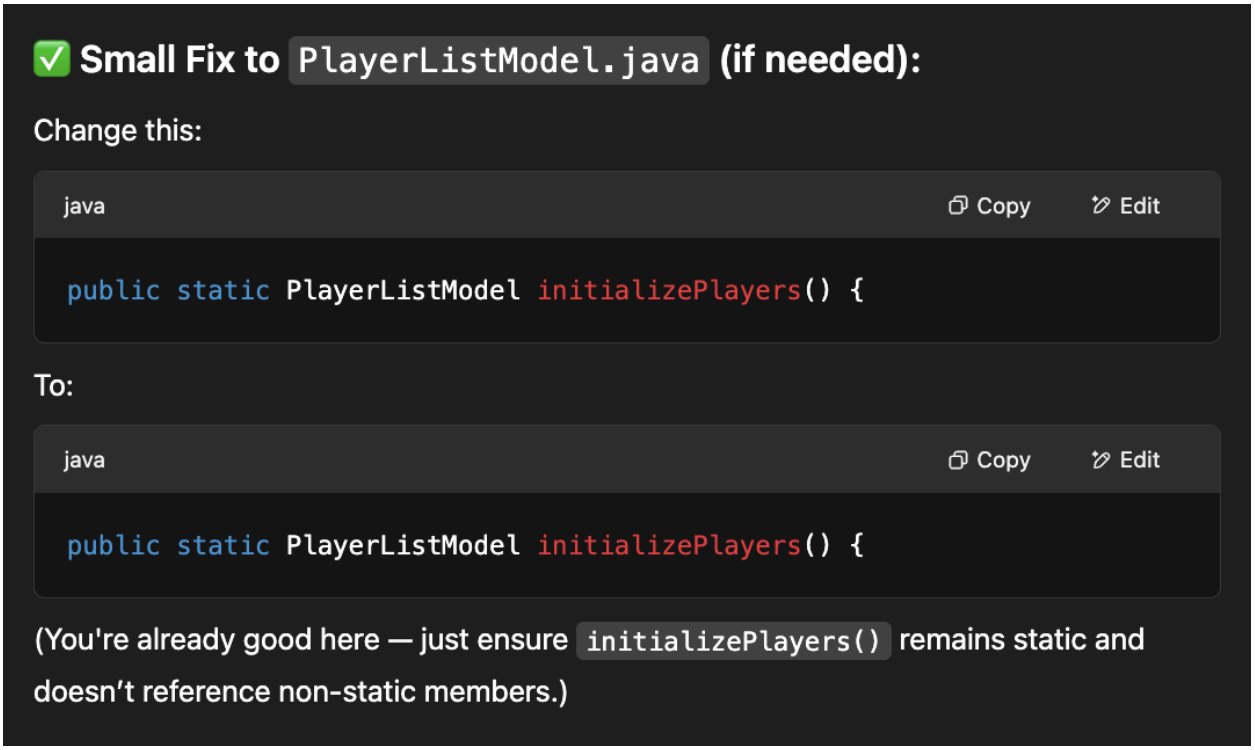
Now I knew I needed to start over, because ChatGPT had completely forgotten about the classes I had provided at the start of our conversation, even though I was only three prompts in and using the newest version of GPT.

After a full hour of pointless back and forth, asking ChatGPT to fix something, causing new issues, asking it to fix those issues, and so on… I decided to forego the AI and just implement it myself to the best of my ability.

I started by moving the game logic methods nextTurn and endTurn from BulldogGameGUI to Referee. However, this broke a lot of things, and fixing those things led me down multiple long paths where every fix brought up multiple more errors. The implementation would require me to completely rewrite the program, which would take time ON TOP OF the time I had already spent trying to manually rework the entire program's logic.

I kept refactoring the classes, changing method parameters, visibilities, reordering method calls, just to try to eliminate the error messages in the code.

I finally had it working again, so I decided to reinvolve the AI to help me clean it up a little. Interestingly enough, in its response, it said something that made absolutely no sense:



✅ **Small Fix to `PlayerListModel.java` (if needed):**

Change this:

```java
public static PlayerListModel initializePlayers() {
```

To:

```java
public static PlayerListModel initializePlayers() {
```

(You're already good here — just ensure `initializePlayers()` remains static and doesn't reference non-static members.)

As you can see, the before/after are exactly the same. And that's not to say that they have the same functionality or they evaluate to the same thing; the line it suggested I change was the same line, character for character, as the line it wanted me to change it to. The code it gave me also did not work. Therefore I did not fully integrate GameStatus into the program, because it would have taken me all day.

Total task time: 4 hours

Task 2: Refactoring the play() method in Player.java

I asked the AI to rewrite the Player class to implement the play() method and then it gave some suggestions on how to integrate the new abstract continueTurn() method into each of the individual player subclasses. The AI gave me some odd variable placements that I had to replace with calls to Referee.turnScore and Referee.checkWinCondition(). This was probably also due to the unpolished game logic from my attempt at integrating a GameStatus class.

Ultimately, the AI was not very helpful on this task and it was much faster to do it myself. Even still, it took a few rewrites to get the interaction between play() and continueTurn() to work. To test this original implementation, I rewrote FifteenPlayer.java by removing play() and implementing continueTurn().

Note that the method needed to have a longer name like continueTurn() because the word “continue” is a Java keyword and therefore my IDE did not allow it as a method name.

Total task time: 1 hour

Task 3: Implementing continueTurn() in Player subclasses

Just like in the previous task, AI was not helpful for this task. I ended up doing the entire implementation myself.

Here is the play() method and the implemented continueTurn() methods:

play() in Player.java:

The “rollRecord” String is to store the numbers rolled so that I can display it on the GUI after a nonhuman player’s turn; this way the user(s) can see what the computer player rolled and see how they got the score they got.

```
/**
 * Plays a turn by rolling dice until the strategy decides to stop,
 * or the player rolls a 6 and loses the turn.
 *
 * @param dice The dice object to roll.
 * @return The score earned during this turn.
 */
public String play(Dice dice) {
    String rollRecord = "    ";
    while (true) {
        int roll = dice.roll();
        rollRecord += roll + " ";
        if (roll == 6) {
           Referee.turnScore = 0;
           break;
        }
        Referee.turnScore += roll;
        Referee.checkWinCondition();
        if (!continueTurn(Referee.turnScore)) {
            break;
        }}
    return rollRecord;
}
```

FifteenPlayer:

```
/**
 * Plays a turn where the player stops rolling when their score reaches or exceeds 15.
 *
 * @param gameDice The dice used for rolling.
 * @return The score earned during the turn.
 */
@Override
protected boolean continueTurn(int turnScore) {
    if (turnScore >= 15)
        return false;
    return true;
}
```

HumanPlayer:

This implementation was odd because the actual game logic for HumanPlayer is moved to Referee, where HumanPlayer is controlled through JSwing buttons.

```
/**
 * Plays a turn where the player rolls the dice and decides whether to continue.
 * @param gameDice The dice used for rolling.
 * @return The score earned during the turn.
 */
@Override
protected boolean continueTurn(int turnScore) {
    return false;
}
```

RandomPlayer:

```
/**
 * Plays a turn where the player randomly decides to roll again or stop,
 * using a shared Random instance via a 2-sided Dice object.
 * @param gameDice The 6-sided dice used for rolling scores.
 * @return The score earned during the turn.
 */
@Override
protected boolean continueTurn(int turnScore) {
    RandomDice randomDecision = new Dice(2);
    if (randomDecision.roll() == 1)
        return false;
    return true;
}
```

UniquePlayer:

```
private int uniquePlayDecider = 0;
private RandomDice uniquePlayDeciderDie = new Dice(2);
/**
 * Plays a turn where the player rolls at least 2 times, then has a 50% chance to continue rolling.
 * @param gameDice The dice used for rolling.
 * @return The score earned during the turn.
 */
@Override
protected boolean continueTurn(int turnScore) {
    uniquePlayDecider++;
    if (uniquePlayDecider >= 2) {
        if (uniquePlayDeciderDie.roll() == 1) {
            uniquePlayDecider = 0;
            return true;
        }}
    uniquePlayDecider = 0;
    return false;
}
```

WimpPlayer:

```
  /**
   * Plays a turn where the player rolls once and stops immediately.
   *
   * @param gameDice The dice used for rolling.
   * @return The score earned during the turn.
   */
  @Override
  protected boolean continueTurn(int turnScore) {
     return false;
  }
```

Total task time: 30 minutes

Task 4: the JavaDoc

Finally, a task suitable for ChatGPT. Luckily, most of the code was already commented. Most of the Javadoc was done already because there were no new classes and very few new methods; for example, in the Player classes, the Javadoc stayed mostly the same because the logic for the continueTurn() method followed the same logic that the play() method had. ChatGPT only had to

Total task time (with proof-reading)**:** 20 minutes